# PSS Q-learning: Improving Q-learning by Probability States Search

Zhanghan Ke(55460880), Aoyun Zhang(55590477)

{zhanghake2-c, aoyuzhang2-c}@my.cityu.edu.hk

**Abstract.** We propose a probability states search (PSS) method to improve the standard Q-learning. Q-learning is a powerful method for Reinforcement Learning and it achieves great result in many tasks. However, this method converges slowly due to the random $\epsilon$-greedy and the unreliable Q-table in the early stage. Our PSS Q-learning uses a probability distribution of "right" records as prior to help the agent choose next action selection strategy. This improved version speeds up the convergence rate of Q-learning remarkably.

## 1 Introduction

Reinforcement Learning (RL) is an important branch of machine learning, it has many applications in the fields of analysis and prediction. In Reinforcement Learning, an agent can observe the environment and execute actions in the environment. For each action, the agent gets the corresponding environment states and the reward of this action. The objective of RL is to get a strategy to take actions according to states and maximize cumulative reward.

One power method of RL is Q-learning, an value-based learning algorithm to train an agent to play the game in a certain environment by a Q-table. In the learning process, an action is chosen in the current state to observe the outcome state and reward after taking the action, then the Q-table is updated by the reward. By this way, agent could learn how to gain a higher score in the game. But one main problem in Q-learning is that it converges very slowly. There are two parts of the algorithm may cause it. First, the default action selection strategy of Q-learning is $\epsilon$-greedy, which selects action randomly at a ratio of  to balance between exploration and exploitation. However, the state with random exploration may be recorded before, therefore the $\epsilon$-greedy will explores some repeated states, which is very ineffective for searching. Second, at the beginning stage, Q-table be updated by unreliable value since the agent is unable to choose the right action, it also causes the problem for convergence.

To address above problems, we use probability estimation to determine whether a state need to be explored or only used history Q-value, and also try to improve the action selection strategy. We verify our improved algorithm in a famous game, *Flappy Bird*, to show the performance.

## 2 Related Work

### 2.1 Gaussian Mixture Model

A Gaussian Mixture Model (GMM) [1] is a parametric probability density function represented as a weighted sum of Gaussian component densities and can be represented as a weighted sum of M component Gaussian densities as:

$$p(x|\lambda) = \sum_{i=1}^{n} w_i g(x|\omega_i, \Sigma_i) \tag{1}$$

where $x$ is the input data, $w_i$ is the mixture weight and g is the component density. Each component density is a D-variate Gaussian function of the form:

$$g(x|\omega_i, \Sigma_i) = \frac{1}{(2\pi)^{D/2}|\Sigma_i|^{1/2}} exp\left\{-\frac{1}{2}(x-\mu_i)'\Sigma_i^{-1}(x-\mu_i)\right\} \tag{2}$$

GMM is an extension of the single Gaussian probability density function and can smoothly approximate the density distribution of any shape. As GMM has multiple models, the division will be more elaborate and suitable for multi-category division.

### 2.2 Q-learning

Q-learning [2] is a form of model-free RL. It provides agents with the capability of learning to act optimally in Markovian domains by experiencing the consequences of actions, without requiring them to build maps of the domains. This algorithm initializes a Q-table with Q-values arbitrarily for all state-action pairs at first. Then in the learning process, the agents choose an action in current state based on Q-values, observe the outcome state and reward after taking the action, and use the result to update Q-table. For a policy $\pi$, Q values are defined as,

$$Q^{\pi}(x,a) = \Re_x(\pi(x)) + \gamma \sum_{y} P_{xy}[\pi(x)]V^{\pi}(y) \tag{3}$$

In the update process of $n^{th}$ episode, the agent adjusts its Q-Values using a learning factor $\alpha_n$, according to,

$$Q_n(x,a) = \begin{cases} (1-\alpha_n)Q_{n-1}(x,a) + \alpha_n[r_n + \gamma V_{n-1}(y_n)] & if\ x = x_n\ and\ a = a_n, \\ Q_{n-1}(x,a) & otherwise, \end{cases} \tag{4}$$

where

$$V_{n-1}(y_n) = \max_b\ Q_(n-1)(y,b)) \tag{5}$$

By this way, Q-learning can learn a policy, which tells an agent what action to take under a certain state.

# 3 Our Method

## 3.1 Overview

In this section, We introduce PSS Q-learning, an improved algorithm using probability distribution to assist state space search. To solve the problem of slow convergence rate caused by $\epsilon$-greedy algorithm and inaccurate Q-table in Q-learning, we use a probability-based search strategy to replace the origin random search strategy.

In our method, we first record some "right" historical records and predict a probability distribution $P$. The "right" means if the agent follows the corresponding action of the recorded state, it will always get positive reward. Then we can use the probability distribution $P$ to predict the most likely action of the agent in each new state. If the prediction result indicates that the probability of a certain action is very high, we could choose the action from Q-table directly, otherwise we will choose an action randomly. In this way, the states in random search strategy are more likely to have not been explored before, and the states with high predicted probability, which means these states have been explored before, can directly use Q-table to speed up the update.

From above, there are two problems need to be resolved. The first problem is how to choose the probability distribution $P$ for state prediction. The second problem is how to get the "right" records used for estimating $P$. That's denoted the agent have an action set with n action, $A \in \{a_1, a_2, \ldots, a_n\}$, and state $s_i$ means the state of agent in frame $i$. We have a Q-table and each Q value is written as $Q(s_i, a_k)$, where $a_k$ is a special action with index $k \in \{1, 2, \ldots, n\}$.

## 3.2 Prior Estimation of State

In our method, the probability distribution $P$ estimated from the historical records is used as a prior. Through this prior, we can determine whether a state $s_i$ need to choose action randomly or use the history value of Q-tabel. We use GMM as the probabilistic model of $P$, and set the number of actions of the agent as the number of components in GMM, then each component of the probability $P_m$ represents the probability of corresponding action. If the computed probability of one action higher than the threshold $\gamma$ in one components, it means in a high probability this state is known, we needn't to random choose action anymore. Otherwise, we need explore the state randomly. Threshold is a hyperparameter that controls the probability of random search. A large threshold require the agent to perform more random search, which will find more unknown states faster, but at the same time, the update of Q-table will be slowed down. On the contrary, a small Threshold update the Q table more quickly, but the speed of exploring new states will be reduced. The Threshold needs to be selected according to the actual problem case by case. The action selection strategy is determined by the GMM prior described above can be expressed as:

$$strategy = \begin{cases} \text{choose action by Q-table}, & \text{if } \arg\max_{P_m} P_m(a_k) > \gamma, \\ \text{choose action randomly}, & \text{otherwise.} \end{cases} \tag{6}$$

We also try to use the estimated $P_m$ as a prior for the values in the weighted Q-table, and we hope to get a more accurate Q value for comparison in this way. However, in practice, this operation has no obvious improvement in performance, so we don't adopt it finally.

### 3.3 Great Round of Game

Another problem in our method is how to get the right replay datset $D$ for probability estimation. In Reinforcement Learning, there is no clear definition of positive and negative samples. The reward obtained by the agent is used for optimization. And since the change of environment is a continuous process, obtaining a positive reward at a certain state does not mean that the performed action is optimal.Therefore, a frame with positive reward can't simply be treated as a "right" record. To address this problem, we define a great round for the game which have a fixed time interval to judge the game, e.g. in *Flappy Bird*, interval between two pipes could be a round, and the great round means bird is not died in this round, so the frames in great round must be right. We can store data of these great round as $D$ to train our model. However, if the game don't have fixed interval, the only way to get D is playing the game for several times by human.

## 4 Experiments

We verify PSS Q-learning on the game *Flappy Bird*, a well-known 2D side-scrolling game. Players control a flying bird to move continuously to the right between pipes. players will lose if the bird touches the pipes. In the game, as shown in figure, we extract three features to combine the state of current frame, including the horizontal distance $x$ and the vertical distance $y$ from the bird to next pipe, and the acceleration of the bird in the direction of gravity $v$, i.e., $s_i = \{x_i, y_i, v_i\}$ as Fig. 1.

The bird has two actions, flying and not flying, i.e. $A = \{a_{fly}, a_{idle}\}$. So The probability distribution of this game is the GMM with the number of components n=2. In the experiment, we set $\gamma$=0.9 and form dataset $D$ by the great rounds in first 1000 games to estimate $P$. We evaluate our algorithm mainly from two aspects: convergence speed and final performance. In each evaluation, we compare original Q-learning, no $\epsilon$-greedy Q-learning and PSS Q-learning.

### 4.1 Convergence Rate Verification

Our objective is to solve the problem of slow convergence in Q-learning, so we first verify that our method does accelerate convergence. We trained all three Q-learning versions in a relatively short time, i.e. playing fewer episodes than normal, and compare the convergence by the test mark. The Fig. 2 shows the convergence results of three Q-learning after 10k games. The blue dot is the score for each game, and the yellow line is the average score change for every 1k game.

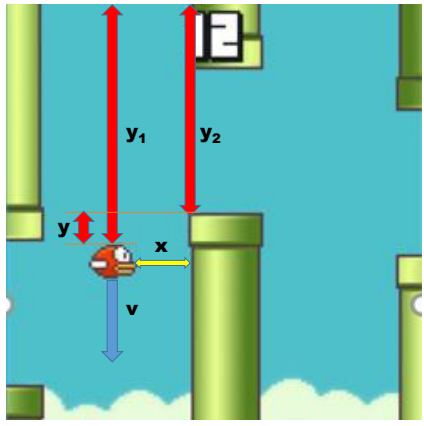
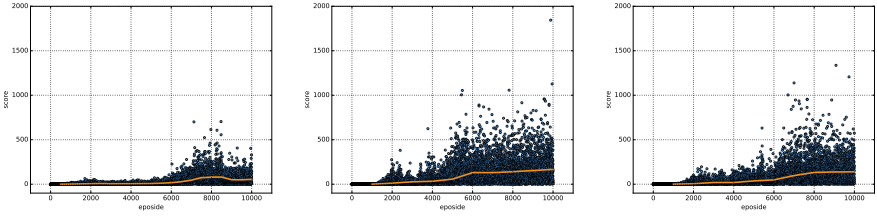



**Fig. 1.** The game screen and the features are used for training.




(a) original Q-learning (b) PSS Q-learning (c) no $\epsilon$-greedy Q-learning

**Fig. 2.** Game results after 10k games. The blue dots are scores of game, and the yellow lines are average scores. Our PSS Q-learning converge quicker than other versions.

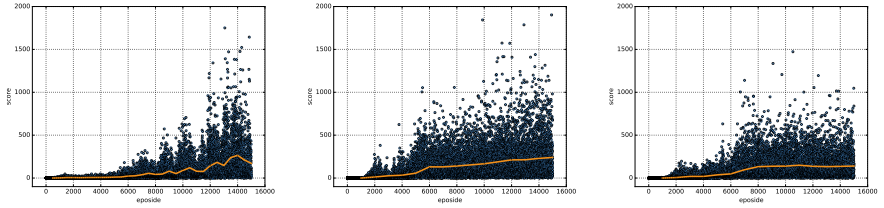
We can find that our method can greatly accelerate the convergence speed of Q-learning algorithm. Even the no $\epsilon$-greedy version also speed up convergence, but our method converge more quicker.

### 4.2 Final Performance Verification

We also train original Q-learningno $\epsilon$-greedy Q-learning and our improved version in enough time. i.e. playing enough episodes to make sure both algorithms are converged, and compare the final performance by the test mark. Fig. 3 shows the results of our experiments after 15k results. The results show PSS Q-learning could gains comparable result than original version. But no $\epsilon$-greedy version is trapped in the local minimum, which cause a lower score

### 4.3 Effective of Threshold

In our method, the hyperparameter threshold $\gamma$ controls the strength of the random state search. As analyzed above, an inappropriate $\gamma$ would cause our method to get a bad result. We do experiments under different $\gamma$ value and note PSS Q-learning is sensitive for this value, 0.9 is a pretty good value for it.




(a) original Q-learning (b) PSS Q-learning (c) no $\epsilon$-greedy Q-learning

**Fig. 3.** Game results after 15k games. All three versions are converged. Our PSS Q-learning could gain comparable results as original Q-learning.

## 5 Conclusions

In this project, we propose PSS Q-learning, a novel version of Q-learning is improved by a GMM. Our also find a way to gain "right" record in RL task, which must help for similar tasks. Our PSS Q-learning speeds up the convergence rate of Q-learning. In addition, it solves the accuracy decline problem of no $\epsilon$-greedy Q-learning and achieves comparable results as original Q-learning.

## References

1. Reynolds, D.: Gaussian mixture models. Encyclopedia of biometrics (2015) 827–832
2. Watkins, C.J., Dayan, P.: Q-learning. Machine learning **8**(3-4) (1992) 279–292